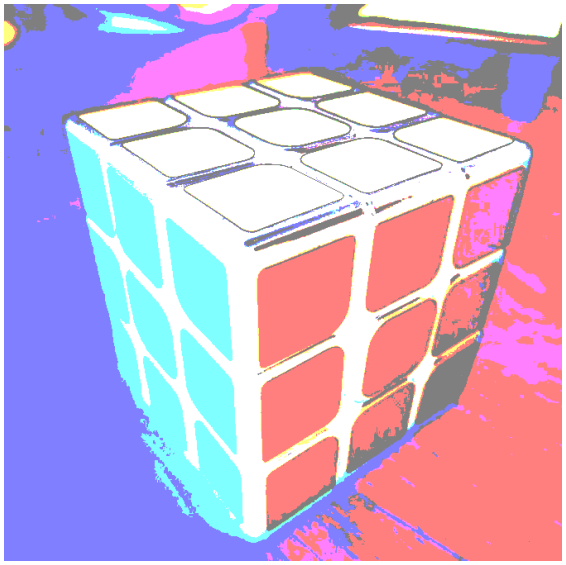
In project 2, I implemented all the required features and some extra features. I implemented **lossy compression and ordered dither**.

Art contest (accident art)



Lossy Compression:

I created a new file format called .zyj This file is similar to ppm, but improve the efficiency. When I implemented ordered dither, I found in some images, numbers are listed in sequence. For example, there may be ten 255 in a row. If we represented this as " 255 10", it will save a lot of memory. Therefore, for any image user passed in, I firstly called ordered dither function, then compressed the image as above. If we wrote the sample image as ppm, it would cost 2571kb. But if we use my .zyj format, it will only cost 1746kb.

Difficulty and success

For the first 5 features, I copied from hw2. I implemented contrast and saturation, but the image didn't look right. Later on, I found it was the problem of the value I passed in. It took me a long time to write the blur and sharpen function. I first wrote a function to generate gaussian filter based on mask size. I accidently found the filter is similar to a famous sequence--Pascal's Triangle.

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1

Then the rest of code takes this filter and converts pixel value.

For the dither, I took me a while to understand the concept. After reviewing the lecture and slides, I finished this part. I thought the toughest part for me is the rotation. I understand if we multiply the rotation matrix, we will get a rotation version. But in real implementation, the image rotates around one corner instead of the center. Also, the final image size is confused. I struggled for a few hours and couldn't make it work. So, I went to professor's office hour for seeking help. After repositioning the image to center before

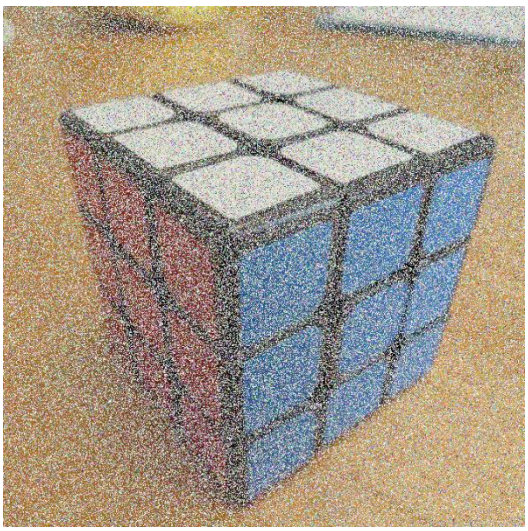
rotation, the rotation matrix works fine and I solve this finally.

I changed the image.h a little bit for pixels around the edge. When we get pixel which are out of boundary, we may take the nearest edge (Gaussian sample) or set it to background (rotate). Therefore, I wrote two separate functions for these two conditions.

Below are some result images.

Noise

.\image -input cube.jpg -noise 0.5 -output result.png

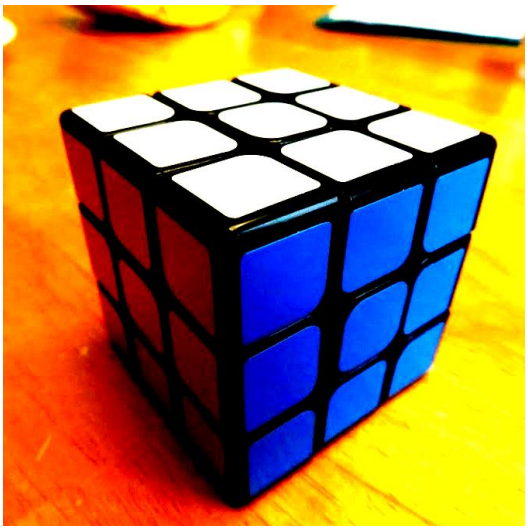


Brightness

.\image -input cube.jpg -brightness 1.5 -output result.png

Contrast

.\image -input cube.jpg -contrast 4 -output result.png



Saturation

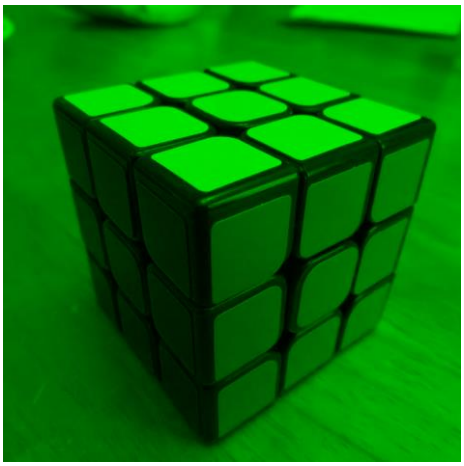
.\image -input cube.jpg -saturation 5 -output result.png

Crop

.\image -input cube.jpg -crop 0 0 200 200 -output result.png

extractChannel

.\image -input cube.jpg -extractChannel 1 -output result.png

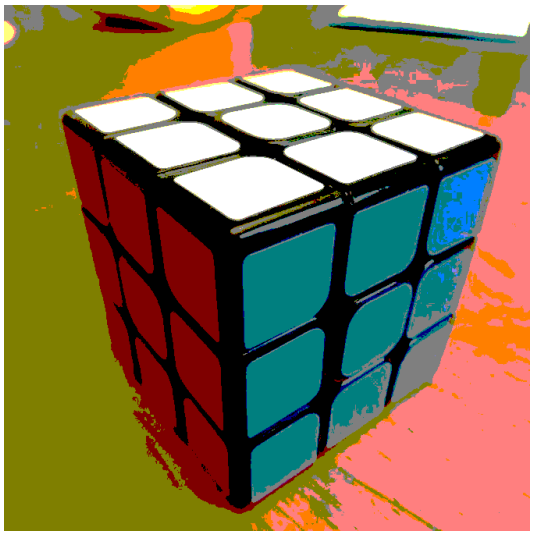


Quantize

.\image -input cube.jpg -quantize 1 -output result.png

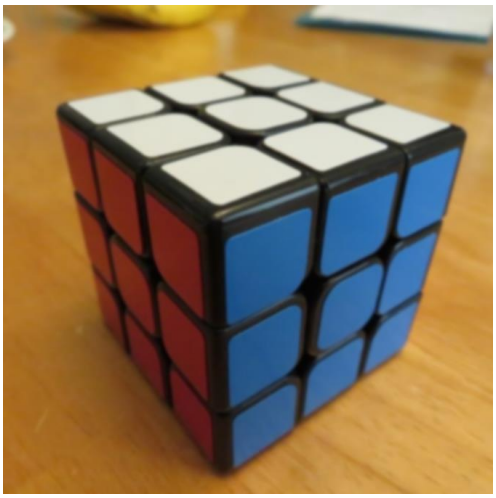
randomDither

.\image -input cube.jpg -randomDither 1 -output result.png



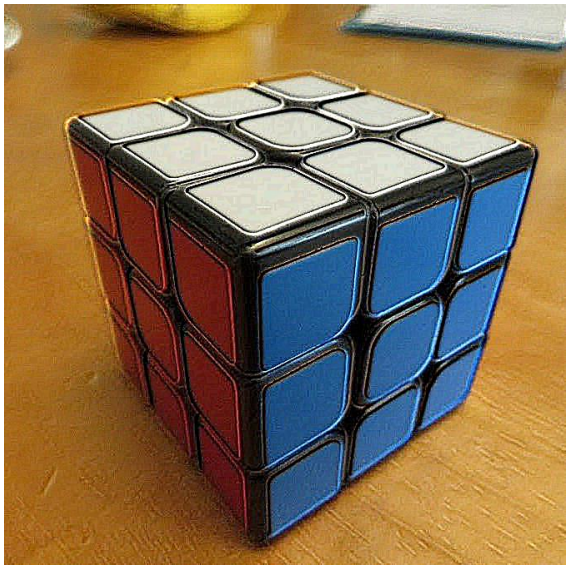
Blur

.\image -input cube.jpg -blur 13 -output result.png



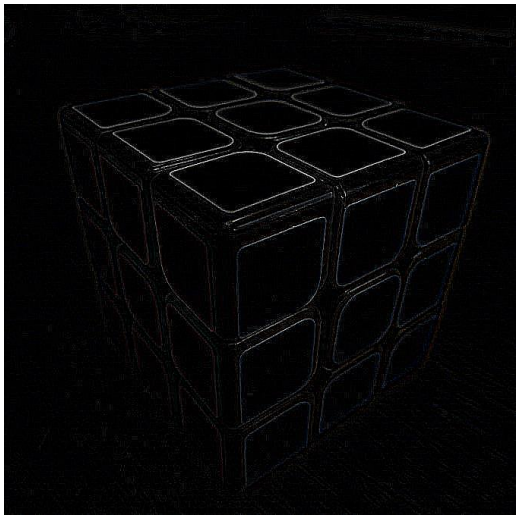
Sharpen

.\image -input cube.jpg -sharpen 10 -output result.png



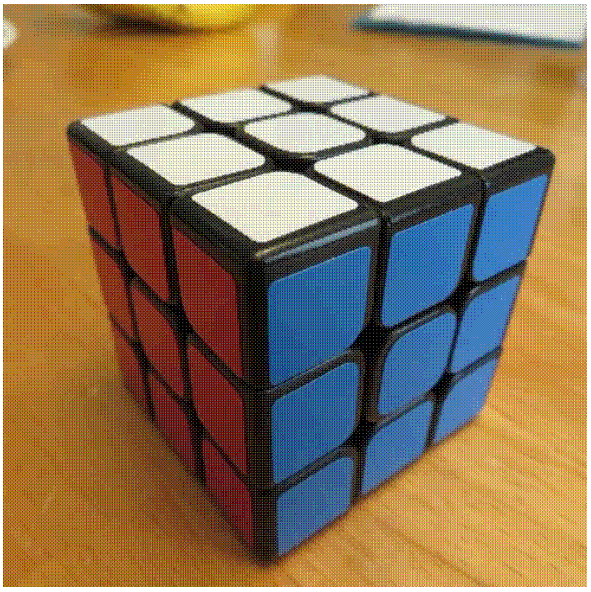
edgeDetect

.\image -input cube.jpg -edgeDetect -output result.png



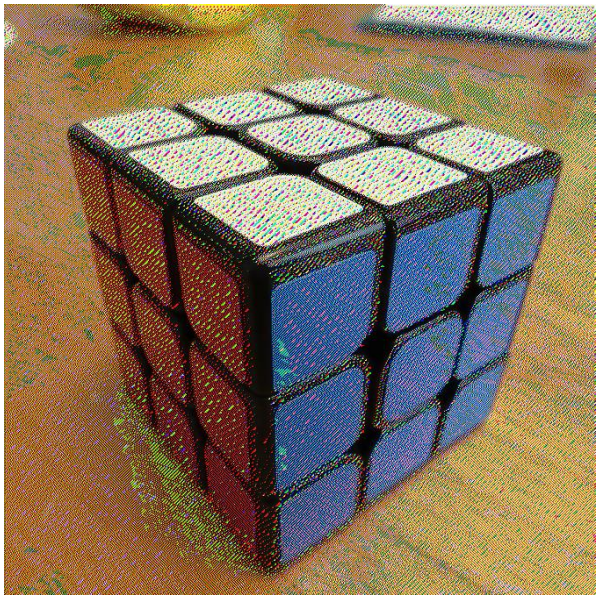
orderedDither

.\image -input cube.jpg -orderedDither 1 -output result.png



FloydSteinbergDither

.\image -input cube.jpg -FloydSteinbergDither 1 -output result.png



Scale

.\image -input cube.jpg -scale 500 200 -output result.png

Rotate

.\image -input cube.jpg -rotate 45 -output result.png

Three sampling functions for scale:

IMAGE_SAMPLING_POINT

IMAGE_SAMPLING_BILINEAR

IMAGE_SAMPLING_GAUSSIAN